Phoenix Pipeline Documentation

Release .1

Open Event Data Alliance

July 14, 2014

Contents

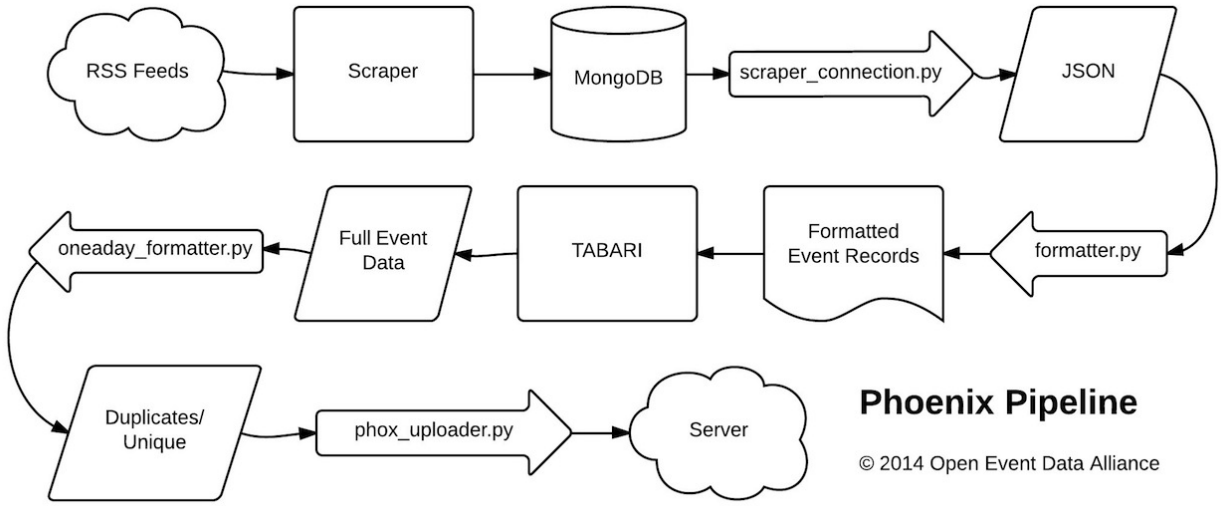
Turning news into events since 2014.

Welcome to the documentation for the Phoenix (PHOX) Pipeline! The PHOX pipeline is a system that links a series of Python programs to convert files from a whitelist of RSS feeds into event data and uploads the data into a designated server. This reference provides a description of how the pipeline works and what it does. It is also written as a programming reference including necessary details on packages, modules, and classes needed to contribute to the codebase.

CHAPTER 1

How it Works

The PHOX pipeline links a series of Python programs to convert files scrapped from a whitelist of RSS feeds to machine-coded event data using the PETRARCH event data coding software. The generated event data is then uploaded to a server designated in a config file. The system is designed to process a single days worth of information that can be included in multiple text files. Below is a flowchart of the pipeline:



Source code can be found at: https://github.com/openeventdata/phoenix_pipeline

Contents:

1.1 Pipeline Details

1.1.1 Configuration File

PHOX_config.ini configures the initial settings for PHOX pipeline and should be included in the working directory.

```
[Server]
server_name = <server name for http: site>
username =  <user name for ftp login to server_name>
password =  <user password for ftp login to server_name>
server_dir = <path to directory on the server where subdirectories are located>
```

```
[Pipeline]
scraper_stem =  <stem for scrapped output>
recordfile_stem =  <stem for output of monger_formatter.py>
fullfile_stem =  <stem for output of TABARI.0.8.4b1>
eventfile_stem =  <stem for event output of oneaday_formatter.py>
dupfile_stem =  <stem for duplicate file output of oneaday_formatter.py>
outputfile_stem =  <stem for files uploaded by phox_uploader.py>
```

Example of PHOX_config.ini

```
[Server]
server_name = openeventdata.org
username = myusername
password = myweakpassword12345
server_dir = public_html/datasets/phoenix/

[Pipeline]
scraper_stem = scraper_results_20
recordfile_stem = eventrecords.
fullfile_stem = events.full.
eventfile_stem = Phoenix.events.
dupfile_stem = Phoenix.dupindex.
outputfile_stem = Phoenix.events.20
```

1.1.2 Web Sources

It is now possible to code event data from a limited list of sources that is different from that used within the web scraper. For instance, it might be desirable to scrape content from a wide variety of sources, but some of this content may be too noisy to include in an event dataset or there is some experementation necessary to determine which sources to include in a final dataset. The data sources are restricted using the `source_keys.txt` file. These keys correspond to those found in the `source` field within the MongoDB instance created by the web scraper.

1.1.3 PETRARCH

PETRARCH (Python Engine for Text Resolution And Related Coding Hierarchy) is an event coding program used to machine code even data from formatted source texts in the pipeline. PETRARCH is the next-generation replacement for the TABARI event data coding software. PETRARCH is dictionary-based and relies on a full parse generated by natural language processing software such as Stanford's CoreNLP along with pattern recognition to identify 'who-did-what-to-whom' relations.

1.2 Contributing Code

You can check out the latest version of the Phoenix Pipeline by cloning this repository using git.

```
git clone https://github.com/openeventdata/phoenix_pipeline.git
```

To contribute to the phoenix pipeline you should fork the repository, create a branch, add to or edit code, push your new branch to your fork of the phoenix pipeline on GitHub, and then issue a pull request. See the example below:

```
git clone https://github.com/YOUR_USERNAME/phoenix_pipeline.git
git checkout -b my_feature
git add... # stage the files you modified or added
```

```
git commit... # commit the modified or added files
git push origin my_feature
```

Commit messages should first be a line, no longer than 80 characters, that summarizes what the commit does. Then there should be a space, followed by a longer description of the changes contained in the commit. Since these comments are tied specifically to the code they refer to (and cannot be out of date) please be detailed.

Note that `origin` (if you are cloning the forked the phoenix pipeline repository to your local machine) refers to that fork on GitHub, *not* the original (upstream) repository https://github.com/openeventdata/ phoenix_pipeline.git. If the upstream repository has changed since you forked and cloned it you can set an upstream remote:

```
git remote add upstream https://github.com/eventdata/phoenix_piepline.git
```

You can then pull changes from the upstream repository and rebasing against the desired branch (in this example, development). You should always issue pull requests against the development branch.

```
git fetch upstream
git rebase upstream/development
```

More detailed information on the use of git can be found in the git documentation.

1.2.1 Coding Guidelines

The following are some guidelines on how new code should be written. Of course, there are special cases and there will be exceptions to these rules. However, following these rules when submitting new code makes the review easier so new code can be integrated in less time.

Uniformly formatted code makes it easier to share code ownership. The petrarch project tries to closely follow the official Python guidelines detailed in PEP8 that detail how code should be formatted and indented. Please read it and follow it.

In addition, we add the following guidelines:

- Use underscores to separate words in non-class names: n_samples rather than nsamples.
- Avoid multiple statements on one line. Prefer a line return after a control flow statement (if/for).
- Use relative imports for references inside petrarch.
- Please don't use `import *`. It is considered harmful by the official Python recommendations. It makes the code harder to read as the origin of symbols is no longer explicitly referenced, but most important, it prevents using a static analysis tool like pyflakes to automatically find bugs in petrarch. Use the numpy docstring standard in all your docstrings.

These docs draw heavily on the contributing guidelines for scikit-learn.

1.3 Phoenix Pipeline Package

1.3.1 `scraper_connection` Module

Downloads scraped stories from Mongo DB.

`scraper_connection`.**`main`**(*current_date*, *file_details*, *write_file=False*, *file_stem=None*)

Function to create a connection to a MongoDB instance, query for a given day's results, optionally write the results to a file, and return the results.

Parameters **current_date: datetime object.** :

Date for which records are pulled. Normally this is $date_running - 1. For example, if the script is running on the 25th, the current_date will be the 24th.

file_details: Named tuple. :

Tuple containing config information.

write_file: Boolean. :

Option indicating whether to write the results from the web scraper to an intermediate file. Defaults to false.

file_stem: String. Optional. :

Optional string defining the file stem for the intermediate file for the scraper results.

Returns posts: Dictionary. :

Dictionary of results from the MongoDB query.

filename: String. :

If *write_file* is True, contains the filename to which the scraper results are writen. Otherwise is an empty string.

`scraper_connection.`**`query_all`**(*collection*, *lt_date*, *gt_date*, *sources*, *write_file=False*)

Function to query the MongoDB instance and obtain results for the desired date range. The query constructed is: greater_than_date > results < less_than_date.

Parameters collection: pymongo.collection.Collection. :

Collection within MongoDB that holds the scraped news stories.

lt_date: Datetime object. :

Date for which results should be older than. For example, if the date running is the 25th, and the desired date is the 24th, then the *lt_date* is the 25th.

gt_date: Datetime object. :

Date for which results should be older than. For example, if the date running is the 25th, and the desired date is the 24th, then the *gt_date* is the 23rd.

sources: List. :

Sources to pull from the MongoDB instance.

write_file: Boolean. :

Option indicating whether to write the results from the web scraper to an intermediate file. Defaults to false.

Returns posts: List. :

List of dictionaries of results from the MongoDB query.

final_out: String. :

If *write_file* is True, this contains a string representation of the query results. Otherwise, contains an empty string.

1.3.2 `formatter` Module

Parses scraped stories from a Mongo DB into PETRARCH-formatted source text input.

formatter.**format_content**(*raw_content*)

Function to process a given news story for further formatting. Calls a function that extract the story text minus the date and source line. Also splits the sentences using the `sentence_segmenter()` function.

Parameters **raw_content: String.** :

Content of a news story as pulled from the web scraping database.

Returns **sent_list: List.** :

List of sentences.

formatter.**get_date**(*result_entry*, *process_date*)

Function to extract date from a story. First checks for a date from the RSS feed itself. Then tries to pull a date from the first two sentences of a story. Finally turns to the date that the story was added to the database. For the dates pulled from the story, the function checks whether the difference is greater than one day from the date that the pipeline is parsing.

Parameters **result_entry: Dictionary.** :

Record of a single result from the web scraper.

process_date: datetime object. :

Datetime object indicating which date the pipeline is processing. Standard is date_running - 1 day.

Returns **date** : String.

Date string in the form YYMMDD.

formatter.**main**(*results*, *file_details*, *process_date*, *thisday*)

Main function to parse results from the web scraper to TABARI-formatted output.

Parameters **results: pymongo.cursor.Cursor. Iterable.** :

Iterable containing the results from the scraper.

file_details: NamedTuple. :

Container generated from the config file specifying file stems and other relevant options.

process_date: String. :

Date for which the pipeline is running. Usually current_date - 1.

this_date: String. :

The current date the pipeline is running.

Returns **new_results: List.** :

List of dictionaries that contain the MongoDB records with new, formatted content.

1.3.3 `oneaday_filter` Module

Deduplication for the final output. Reads in a single day of coded event data, selects first record of souce-target-event combination and records references for any additional events of same source-target-event combination.

oneaday_filter.**filter_events**(*results*)

Filters out duplicate events, leaving only one unique (DATE, SOURCE, TARGET, EVENT) tuple per day.

Parameters **results: Dictionary.** :

PETRARCH-formatted results in the {StoryID: [(record), (record)]} format.

Returns **filter_dict: Dictionary.** :

Contains filtered events. Keys are (DATE, SOURCE, TARGET, EVENT) tuples, values are lists of IDs, sources, and issues.

`oneaday_filter.`**`main`**(*results*)

Pulls in the coded results from PETRARCH dictionary in the {StoryID: [(record), (record)]} format and allows only one unique (DATE, SOURCE, TARGET, EVENT) tuple per day. Returns this new, filtered event data.

Parameters **results: Dictionary.** :

PETRARCH-formatted results in the {StoryID: [(record), (record)]} format.

1.3.4 `result_formatter` Module

Puts the PETRARCH-generated event data into a format consistent with other parts of the pipeline so that the events can be further processed by the `postprocess` module.

`result_formatter.`**`filter_events`**(*results*)

Filters out duplicate events, leaving only one unique (DATE, SOURCE, TARGET, EVENT) tuple per day.

Parameters **results: Dictionary.** :

PETRARCH-formatted results in the {StoryID: [(record), (record)]} format.

Returns **formatted_dict: Dictionary.** :

Contains filtered events. Keys are (DATE, SOURCE, TARGET, EVENT, COUNTER) tuples, values are lists of IDs, sources, and issues. The `COUNTER` in the tuple is a hackish workaround since each key has to be unique in the dictionary and the goal is to have every coded event appear event if it's a duplicate. Other code will just ignore this counter.

`result_formatter.`**`main`**(*results*)

Pulls in the coded results from PETRARCH dictionary in the {StoryID: [(record), (record)]} format and converts it into (DATE, SOURCE, TARGET, EVENT, COUNTER) tuple format. The `COUNTER` in the tuple is a hackish workaround since each key has to be unique in the dictionary and the goal is to have every coded event appear event if it's a duplicate. Other code will just ignore this counter. Returns this new, filtered event data.

Parameters **results: Dictionary.** :

PETRARCH-formatted results in the {StoryID: [(record), (record)]} format.

Returns **formatted_dict: Dictionary.** :

Contains filtered events. Keys are (DATE, SOURCE, TARGET, EVENT, COUNTER) tuples, values are lists of IDs, sources, and issues. The `COUNTER` in the tuple is a hackish workaround since each key has to be unique in the dictionary and the goal is to have every coded event appear event if it's a duplicate. Other code will just ignore this counter.

1.3.5 `postprocess` Module

Performs final formatting of the event data and writes events out to a text file.

`postprocess.`**`create_strings`**(*events*)

Formats the event tuples into a string that can be written to a file.close

Parameters **events: Dictionary.** :

Contains filtered events. Keys are (DATE, SOURCE, TARGET, EVENT) tuples, values are lists of IDs, sources, and issues.

Returns event_strings: String. :

Contains tab-separated event entries with

as a line :

delimiter.

postprocess.**main**(*event_dict*, *this_date*, *file_details*)

Pulls in the coded results from PETRARCH dictionary in the {StoryID: [(record), (record)]} format and allows only one unique (DATE, SOURCE, TARGET, EVENT) tuple per day. Returns this new, filtered event data.

Parameters event_dict: Dictionary. :

PETRARCH-formatted results in the {StoryID: [(record), (record)]} format.

this_date: String. :

The current date the pipeline is running.

file_details: NamedTuple. :

Container generated from the config file specifying file stems and other relevant options.

postprocess.**process_actors**(*event*)

Splits out the actor codes into separate fields to enable easier querying/formatting of the data.

Parameters event: Tuple. :

(DATE, SOURCE, TARGET, EVENT) format.

Returns actors: Tuple. :

Tuple containing actor information. Format is (source, source_root, source_agent, source_others, target, target_root, target_agent, target_others). Root is either a country code or one of IGO, NGO, IMG, or MNC. Agent is one of GOV, MIL, REB, OPP, PTY, COP, JUD, SPY, MED, EDU, BUS, CRM, or CVL. The `others` contains all other actor or agent codes.

postprocess.**process_cameo**(*event*)

Provides the "root" CAMEO event, a Goldstein value for the full CAMEO code, and a quad class value.

Parameters event: Tuple. :

(DATE, SOURCE, TARGET, EVENT) format.

Returns root_code: String. :

First two digits of a CAMEO code. Single-digit codes have leading zeros, hence the string format rather than

event_quad: Int. :

Quad class value for a root CAMEO category.

goldstein: Float. :

Goldstein value for the full CAMEO code.

postprocess.**split_process**(*event*)

Splits out the CAMEO code and actor information along with providing conversions between CAMEO codes and quad class and Goldstein values.

Parameters event: Tuple. :

(DATE, SOURCE, TARGET, EVENT) format.

Returns formatted: Tuple. :

Tuple of the form (year, month, day, formatted_date, root_code, event_quad).

actors: Tuple. :

Tuple containing actor information. Format is (source, source_root, source_agent, source_others, target, target_root, target_agent, target_others). Root is either a country code or one of IGO, NGO, IMG, or MNC. Agent is one of GOV, MIL, REB, OPP, PTY, COP, JUD, SPY, MED, EDU, BUS, CRM, or CVL. The `others` contains all other actor or agent codes.

1.3.6 `geolocation` Module

Geolocates the coded event data.

`geolocation.`**`main`**(*events*, *file_details*)

Pulls out a database ID and runs the `query_geotext` function to hit the GeoVista Center's GeoText API and find location information within the sentence.

Parameters events: Dictionary. :

Contains filtered events from the one-a-day filter. Keys are (DATE, SOURCE, TARGET, EVENT) tuples, values are lists of IDs, sources, and issues.

Returns events: Dictionary. :

Same as in the parameter but with the addition of a value that is a tuple of the form (LAT, LON).

`geolocation.`**`query_geotext`**(*sentence*)

Filters out duplicate events, leaving only one unique (DATE, SOURCE, TARGET, EVENT) tuple per day.

Parameters sentence: String. :

Text from which an event was coded.

Returns lat: String. :

Latitude of a location.

lon: String. :

Longitude of a location.

1.3.7 `uploader` Module

Uploads PETRARCH coded event data and duplicate record references to designated server in config file.

`uploader.`**`get_zipped_file`**(*filename*, *dirname*, *connection*)

Downloads the file filename+zip from the subdirectory dirname, reads into tempfile.zip, cds back out to parent directory and unzips Exits on error and raises RuntimeError

`uploader.`**`main`**(*datestr*, *server_info*, *file_info*)

When something goes amiss, various routines will and pass through a RuntimeError(explanation) rather than trying to recover, since this probably means something is either wrong with the ftp connection or the file structure got corrupted. This error is logged but needs to be caught in the calling program.

uploader.**store_zipped_file**(*filename*, *dirname*, *connection*)
Zips and uploads the file filename into the subdirectory dirname, then cd back out to parent directory. Exits on error and raises RuntimeError

1.3.8 utilities Module

Miscellaneous functions to do things like establish database connections, parse config files, and intialize logging.

utilities.**do_RuntimeError**(*st1*, *filename=u''*, *st2=u''*)
This is a general routine for raising the RuntimeError: the reason to make this a separate procedure is to allow the error message information to be specified only once. As long as it isn't caught explicitly, the error appears to propagate out to the calling program, which can deal with it.

utilities.**init_logger**(*logger_filename*)
Initialize a log file.

Parameters **logger_filename: String.** :

Path to the log file.

utilities.**make_conn**(*db_auth*, *db_user*, *db_pass*)
Function to establish a connection to a local MonoDB instance.

Parameters **db_auth: String.** :

MongoDB database that should be used for user authentication.

db_user: String. :

Username for MongoDB authentication.

db_user: String. :

Password for MongoDB authentication.

Returns **collection: pymongo.collection.Collection.** :

Collection within MongoDB that holds the scraped news stories.

utilities.**parse_config**(*config_filename*)
Parse the config file and return relevant information.

Parameters **config_filename: String.** :

Path to config file.

Returns **server_list: Named tuple.** :

Config information specifically related to the remote server for FTP uploading.

file_list: Named tuple. :

All the other config information not in `server_list`.

utilities.**sentence_segmenter**(*paragr*)
Function to break a string 'paragraph' into a list of sentences based on the following rules:

1.Look for terminal [.,?,!] followed by a space and [A-Z]

2. If ., check against abbreviation list ABBREV_LIST: Get the string between the . and the previous blank, lower-case it, and see if it is in the list. Also check for single-letter initials. If true, continue search for terminal punctuation 3. Extend selection to balance (...) and "...". Reapply termination rules 4. Add to sentlist if the length of the string is between MIN_SENTLENGTH and MAX_SENTLENGTH 5. Returns sentlist

Parameters **paragr: String.** :

Content that will be split into constituent sentences.

Returns sentlist: List. :

List of sentences.

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*

f

g

o

p

r

s

u

f

g

o

p

r

s

u